ONR POST TENURE SUPPORT

DEVELOPMENT OF A MASSIVELY PARALLEL PARTICLE-MESH ALGORITHM FOR SIMULATIONS OF GALAXY DYNAMICS AND PLASMAS

Final Report

1996

GMU PROPOSAL NUMBER 093148
GOVERNMENT GRANT NUMBER : N00014-93-1-0193

PI: John Wallin, George Mason University

Introduction

Particle-mesh calculations treat forces and potentials as field quantities which are represented approximately on a mesh. A system of particles is mapped on to this mesh as a density distribution of mass or charge. The Fourier transform is used to convolve this distribution with the Green's function of the potential, and a finite difference scheme is used to calculate the forces acting on the particles. The computation time scales as the $N_g \log N_g$, where N_g is the size of the computational grid. In contrast, the particle-particle method's computing time relies on direct summation, so the time for each calculation is given by N_p^2, where N_p is the number of particles.

The particle-mesh method is best suited for simulations with a fixed minimum resolution and for collisionless systems, while hierarchical tree codes have proven to be superior for collisional systems where two-body interactions are important. Particle mesh methods still dominate in plasma physics where collisionless systems are modeled.

The CM-200 Connection Machine produced by Thinking Machines Corp. is a data parallel system. On this system, the front-end computer controls the timing and execution of the parallel processing units. The programming paradigm is Single-Instruction, Multiple Data (SIMD). The processors on the CM-200 are connected in an N-dimensional hypercube; the largest number of links a message will ever have to make is N. As in all parallel computing, the efficiency of an algorithm is primarily determined by the fraction of the time spent communicating compared to that spent computing. Because of the topology of the processors, nearest neighbor communication is more efficient than general communication.

Program Structure

Under this grant, a parallel version of the particle-mesh algorithm was implemented on the CM-200 Connection Machine at the Naval Research Laboratory. This algorithm was developed in the C* programming language that is designed for easy implementation of SIMD codes. Figure 1 shows a flow chart of this algorithm.

In PM codes, there are two fundamental computational units, namely the computational grid and the array of particles. This program represented each of these two computational units as processor shapes. The C* language groups sets of virtual processors - data units which have common characteristics- into user defined processor shapes. These shape can be added as variables, or directly and indirectly addressed.

The grid in PM codes is used to calculate the potential and the forces, while the particles are moved according to the forces determined from the grid. Every timestep requires a mapping of the particles to the computational grid, and a mapping of the grid's forces back to the particles (see Figure 2). Since particles can move throughout the computational grid, general communications must be used to connect the particle shape to the grid shape. Global communications must also b used to create the Fourier transform of the particle density and the inverse transform of the potential. Besides the global communications, local communications between neighboring particles must be used for the finite difference scheme to determine the forces from the convolved potentials.

Two schemes were investigated for mapping the particles to the grid. The nearest grid point scheme (NGP) maps each particle into a single cell, while the cloud-in-cell scheme (CIC) maps the density of each particle into four weighted parts and spreads them between cells. The CIC scheme has generally higher accuracy since it creates a linearly interpolated density distribution (see Figure 3). The price of this higher accuracy is the additional time spent mapping the particles to the grid and interpolating the force back to the particles.

Research Results

Timing tests of the PM algorithm show that for simulations with less than 512K particles, the execution time scales linearly with the number of particles. Figure 4 shows the total cpu time per algorithm timestep as a function of the number of particles.

Most of the time in the algorithm is spent in the parts of the code which require general communication between processors. Since the CM-200 is optimized for local communication, the mapping between the particles and the grid is especially time consuming.

Table 1 contains the results from the timing test on the system for five different runs using the NGP method. Communication time begins to dominate the execution time for runs with more than 512K particles. The largest time portion of the communication time is spent calculating the forces on the particles. Since each particle must access the force from grid cells, two way communications must be used in this section of the code. The FFT routine on the CM-200 is very efficient, and only takes up about 2.5 seconds to transform the grid used in this study.

When the CIC scheme was tested, the communications became significantly slower. Mapping the particles to the grid took more than four times longer, because each particle had to be mapped four times. Force calculations and interpolation also increased in complexity. No significant increase in the simulation accuracy was seen because of these changes.

One area that was not investigated is the effects of a non-uniform distribution of particles. If most of the particles must be mapped to a single grid cell and virtual processor, a communications bottleneck will occur in this algorithm. Although this is less important for plasma and cosmology simulations, simulations of interacting galaxies have highly variable densities which can create this bottleneck.

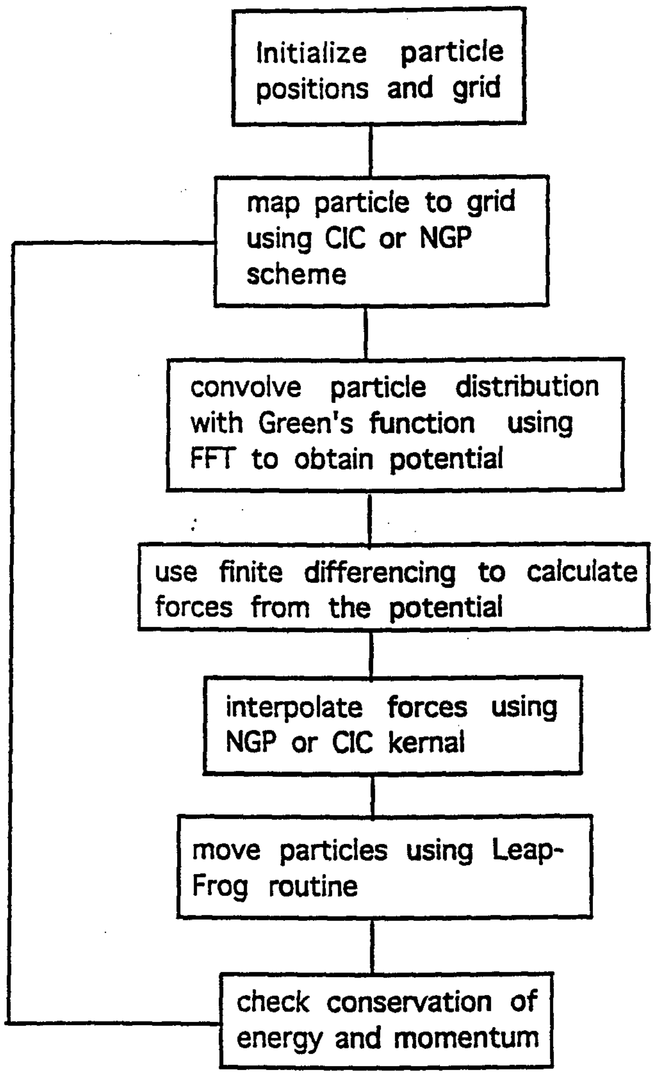
Conclusions

The PM method can be efficiently implemented on a SIMD parallel computer with relatively simple data structures and communications. Fine grain parallelism can be effectively used in this type of algorithm if general communication between processors has been optimized. This type of algorithm will be very effective for plasma and cosmological simulations, which are both collisionless and have relatively uniform densities. No significant increase in accuracy was seen when the CIC method as used, and a significant increase in computational cost was associated with its implementation.

Publications Resulting from this Study

Wallin, J. And Shah, I., *Design and Performance of a Parallel Particle-Mesh Code as a Solution to the N-body Problem,* abstract in the **Bulletin of the American Astronomical Society** (June 1993)

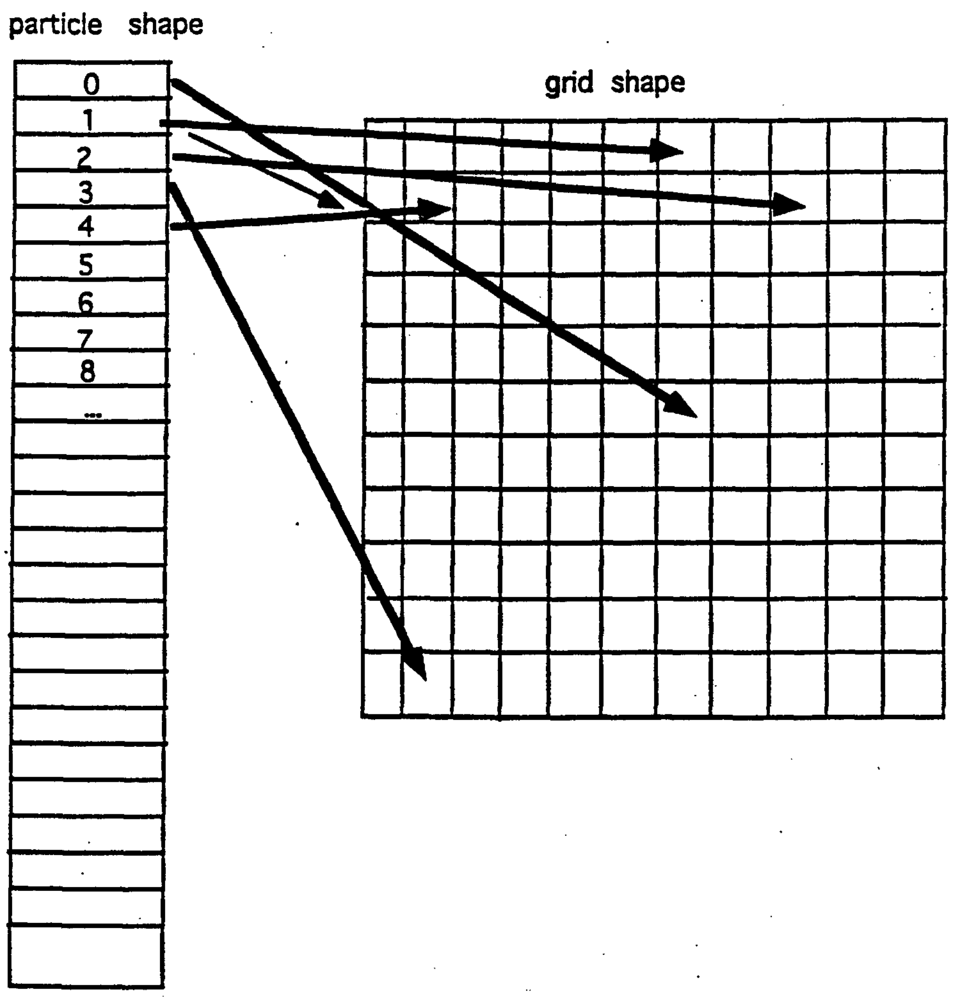
Figure 1 - Flow chart for the parallel particle mesh algorithm


Initialize particle positions and grid
map particle to grid using CIC or NGP scheme
convolve particle distribution with Green's function using FFT to obtain potential
use finite differencing to calculate forces from the potential
interpolate forces using NGP or CIC kernal
move particles using Leap-Frog routine
check conservation of energy and momentum

Flow of control in Particle Mesh Codes

Figure 2 - Mapping the particles to the computational grid

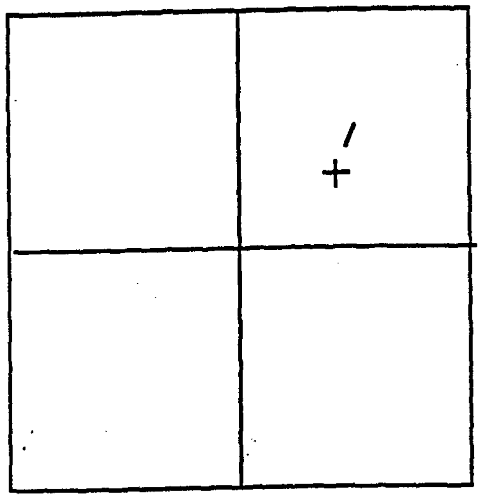
Communication between the particle and grid processor shapes



Communication between the particle and grid processes geometries requires general communication. The same 8k physical processors are used for all computations, but the virtual shapes define the control flow for the data.

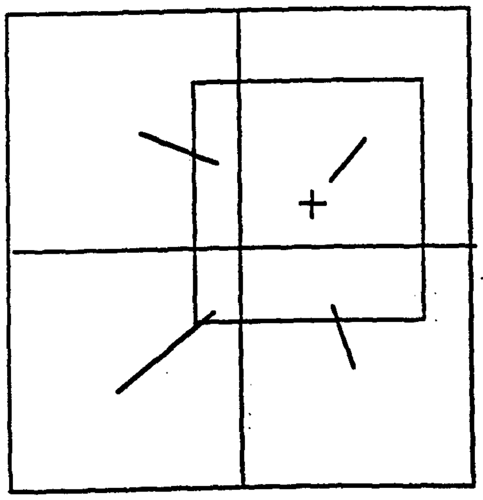
Figure 3 - Assignment Function Geometry

ASSIGNMENT FUNCTION GEOMETRY



NGP METHOD

In the NGP scheme, all the mass of the particle is distributed into the nearest adjoining cell



CIC METHOD

In the CIC scheme, the mass of the particle is distributed into groups of adjoining cells based on gemoetry weighting of the cloud shape

Figure 4 - Timing test results

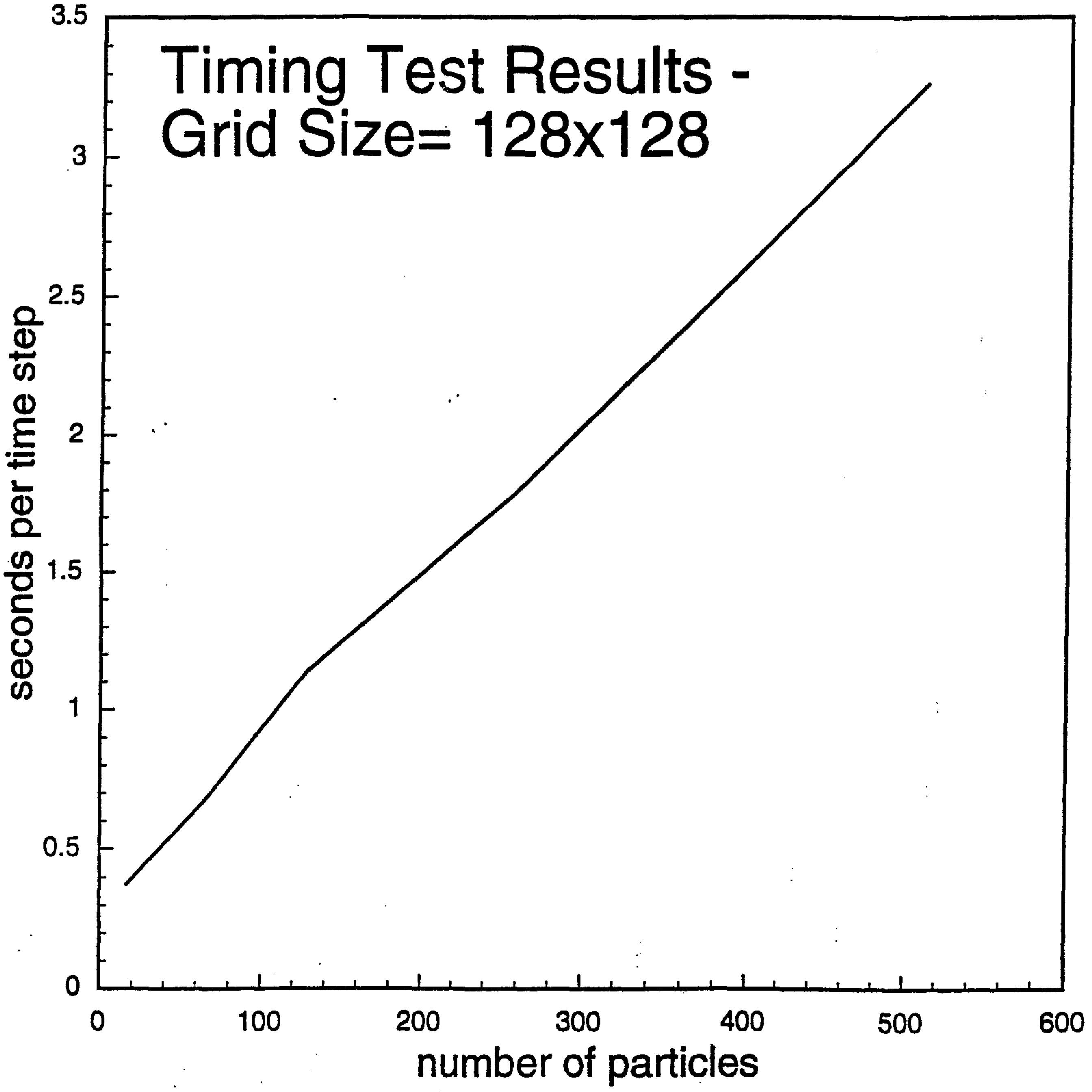


Table 1: Execution time of the particle PM code.

	16K	32K	64K	128K	256K	512K
Front end	10.8	14.4	17.8	24.1	37.0	59.4
Comm	3.6	6.9	12.8	24.3	37.8	75.2
CM CPU	1.6	2.8	5.3	11.4	18.5	35.2
CM total	5.5	10.2	18.2	37.6	59.5	116.4

Each column represents the number of seconds for a single simulation, with the top row indicating the total number of particles in each simulation. The grid was fixed at 128^2 for these simulations. The time on the front end includes time to read data and execute any serial instructions. The Comm row is the total time spent communicating. The CM CPU is the total amount of time executing on the Connection Machine CPU, while CM total is the total cpu and communication time.